대용량 데이터를 위한 전역적 범주화를 이용한 결정 트리의 순차적 생성

한 경 식[†] · 이 수 원[††]

요 약

최근 들어, 대용량의 데이터를 처리할 수 있는 트리 생성 방법에 많은 관심이 집중되고 있다. 그러나 대용량 데이터를 위한 대부분의 알고리즘은 일괄처리 방식으로 데이터를 처리하기 때문에 새로운 데이터가 추가되면 이 데이터를 반영한 결정 트리를 생성하기 위해 처음부터 트리를 다시 생성해야 하다. 이러한 재생성에 따른 비용문제에 보다 효율적인 접근 방법은 결정 트리를 순차적으로 생성하는 접근 방법이다. 대표적인 알고리즘으로 BOAT와 ITI를 들 수 있으며 이들 알고리즘은 수치형 데이터 처리를 위해 지역적 범주화를 이용한다. 그러나 범주화는 정렬된 형태의 수치형 데이터를 요구하기 때문에 대용량 데이터를 처리해야하는 상황에서 전체 데이터에 대해 한번만 정렬을 수행하는 전역적 범주화 기법이 모든 노드에서 매번 정렬을 수행하는 지역적 범주화보다 적합하다. 본 논문은 수치형 데이터 처리를 위해 전역적 범주화를 이용하여 생성된 트리를 효율적으로 재생성하는 순차적 트리 생성 방법을 제안한다. 새로운 데이터가 추가될 경우, 전역적 범주화에 기반 한 트리를 순차적으로 생성하기 위해서는 첫째, 이 새로운 데이터가 반영된 범주를 재생성해야 하며, 둘째, 범주 변화에 맞게 트리의 구조를 변화시켜야한다. 본 논문에서는 효율적인 범주 재생성을 위해 샘플 분할 포인트를 추출하고 이로부터 범주화를 수행하는 기법을 제안하며 범주 변화에 맞는 트리 구조 변화를 위해 신뢰구간과 트리 재구조화기법을 이용한다. 본 논문에서 피플 데이터베이스를 이용하여 기존의 지역적 범주화를 이용한 경우와 비교 실험하였다.

키워드 : 결정 트리, 순차적 학습, 전역적 범주화, 대용량 데이터, 데이터 마이닝

Incremental Generation of A Decision Tree Using Global Discretization For Large Data

Kyong Sik Han[†] · Soo Won Lee[††]

ABSTRACT

Recently, It has focused on decision tree algorithms that can handle large dataset. However, because most of these algorithms for large datasets process data in a batch mode, if new data is added, they have to rebuild the tree from scratch. A more efficient approach to reducing the cost problem of rebuilding is an approach that builds a tree incrementally. Representative algorithms for incremental tree construction methods are BOAT and ITI and most of these algorithms use a local discretization method to handle the numeric data type. However, because a discretization requires sorted numeric data, in situation of processing large data sets, a global discretization method that sorts all data only once is more suitable than a local discretization method that sorts in every node. This paper proposes an incremental tree construction method that efficiently rebuilds a tree using a global discretization method to handle the numeric data type. When new data is added, new categories influenced by the data should be recreated, and then the tree structure should be changed in accordance with category changes. This paper proposes a method that extracts sample points and performs discretization from these sample points to recreate categories efficiently and uses confidence intervals and a tree restructuring method to adjust tree structure to category changes. In this study, an experiment using people database was made to compare the proposed method with the existing one that uses a local discretization.

Key Words : Decision Tree, Incremental Learning, Global Discretization, Large Dataset, Data Mining

1. 서 론

최근 들어, 대용량의 데이터를 처리할 수 있는 트리 생성 방법에 많은 관심이 집중되고 있다. 그 이유는 기존의 메모리 기반 알고리즘은 오늘날의 대용량 데이터를 처리하지 못한다는 한계점과 더 많은 데이터를 이용하여 생성된 결정 트리가 더 좋은 정확도를 보이고 있기 때문이다[1, 2]. 대용량 데이터를 위한 대표적인 결정 트리 알고리즘으로 SLIQ[3], SPRINT[4], RainForest[5] 등이 있다. 그러나 대용

※ 본 연구는 숭실대학교 교내연구비로 지원되어 이루어짐.
† 준 회 원 : (주)인우기술 선임연구원
†† 정 회 원 : 숭실대학교 컴퓨터학부 부교수
논문접수 : 2005년 3월 23일, 심사완료 : 2005년 5월 24일

량 데이터를 위한 대부분의 알고리즘은 일괄처리 방식(Batch Approach)으로 데이터를 처리하기 때문에 새로운 데이터가 추가되면 이 데이터를 반영한 결정 트리를 생성하기 위해 처음부터 트리를 다시 생성해야 하다.

이러한 트리 재생성이 필요한 환경에 보다 효율적인 접근 방법은 결정 트리를 순차적으로 생성하는 접근 방법(Incremental Approach)이다[6, 7]. 새로운 데이터가 추가될 때, 결정 트리를 처음부터 재생성하는 것 보다 기존의 결정 트리를 유지하고 기존의 트리가 이 데이터를 반영할 수 있도록 갱신하는 것이 비용이 훨씬 적게 들기 때문이다. 결정 트리를 순차적으로 생성하는 대표적인 알고리즘으로 BOAT[7]와 ITI[6] 등을 들 수 있다. BOAT는 부트스트랩핑 기법(Bootstrapping)을 이용한 순차적 알고리즘으로서 대용량 데이터를 지원한다. BOAT는 수치형 분할 변수(Split Variable)에 대해 최종 분할 포인트(Final Split Point)가 존재할 가능성이 있는 신뢰구간을 유지하고 있기 때문에 그 범위 안에서 발생하는 분할 포인트 변경에 대해서는 효율적으로 처리하지만 최종 분할 포인트가 그 신뢰구간을 넘어서는 경우와 최종 분할 변수가 변경되면 그에 영향을 받는 부분은 다시 생성해야 한다는 문제점을 안고 있다. 이에 반해, ITI는 분할 포인트 변경과 분할 변수 변경을 효율적으로 처리하지만 대용량 데이터를 처리하지 못해 오늘날의 순차적인 트리 생성 기법으로 적합하지 못하다.

결정 트리를 생성하는 알고리즘은 기본적으로 데이터가 모두 범주형 데이터만으로 구성되었다고 가정하기 때문에 수치형 데이터의 경우, 이를 적절한 범주화 기법(Discretization)을 이용하여 범주형으로 변환한 후 트리를 생성한다. 범주화 기법은 크게 전역적 범주화 기법(Global Discretization)과 지역적 범주화 기법(Local Discretization)으로 분류되며 전역적 범주화 기법은 일반적으로 전처리(Preprocessing)로서 트리 생성 이전에 전체 데이터에 대해 한번 수행되며 지역적 범주화 기법은 트리 생성 과정에서 그 노드에 전달된 데이터만을 이용하는 방법으로 모든 트리 노드에서 수행된다[8]. 범주화는 정렬된 형태의 수치형 데이터를 요구하기 때문에 대용량 데이터를 처리해야하는 상황에서 전처리로서 전체 데이터에 대해 한번만 정렬을 수행하는 전역적 범주화 기법이 모든 노드에서 매번 정렬을 수행하는 지역적 범주화보다 적합하다. 그러나 대부분의 순차적 트리 생성 알고리즘은 범주화기법으로 지역적 범주화를 이용하고 있으며 전역적 범주화를 이용한 순차적 트리 생성 알고리즘에 관한 연구는 없는 상황이다.

기존의 대용량 데이터를 위한 결정트리 알고리즘은 수치형 데이터 처리를 위한 전처리로서 전역적 범주화를 이용할 수 있다. 그러나 새로운 데이터가 추가되어 전역적 범주화 기반 트리를 순차적으로 생성하기 위해서는 첫째, 이 새로운 데이터가 반영된 범주를 재생성해야 하며 둘째, 범주 변화에 맞게 트리의 구조를 변경시켜야한다. 본 논문은 이처럼 전역적 범주화를 사용한 트리에서 새로운 데이터가 추가될 경우 효율적으로 재생성하는 순차적 트리 생성 방법을 제안한다. 본 논문에서는 효율적인 범주 재생성을 위해 샘플 분할 포인트를 추출하고 이로부터 범주화를 수행하는 기법을 제안하며, 범주 변화에 맞는 트리를 재생성하기 위해 신뢰구간과 트리 재구조화기법을 이용하는 방법을 제안한다. 본 논문에서 제안하는 방법은 전처리로서 전역적 범주화를 이용할 경우, BOAT와 같은 다른 순차적 기법에서도 적용 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 범주화 기법과 결정 트리 알고리즘들을 분석하고 3장에서는 순차적으로 범주를 재생성하기 위해 샘플 포인트를 이용하는 순차적 범주 생성 기법을 설명한다. 4장에서는 새로운 데이터 추가로 인해 범주 변화가 발생할 경우, 효율적으로 트리를 재생성하는 방법에 대해 설명한다. 5장에서는 제안하는 접근방법에 대한 대용량 처리 가능성에 대해 실험결과를 기술하고 6장에서 결론 및 향후 연구를 제시한다.

2. 관련 연구

2.1 전역적 범주화 알고리즘

전역적 범주화의 대표적인 기법은 불순도(Impurity)에 기반 한 범주화 기법이다. 불순도 기반 범주화 기법은 Gini Index[9], Information Gain[10], Gain Ratio[10] 등을 이용한 기법을 들 수 있다. 그 중 Information Gain을 이용한 기법은 분할 포인트를 선택하기 위해 후보 분할 포인트(Candidate Split Point)들의 평균 클래스 엔트로피(Average Class Entropy)를 이용하며 정의는 (그림 1)과 같다. IG(A,T;S)와 E(A,T;S)는 각각 Information Gain 함수 및 평균 클래스 엔트로피 함수이며 S는 전체 데이터 집합, A는 수치형 변수, T는 후보 분할 포인트, S1 및 S2는 S를 T로 분할할 경우의 부분 구간을 나타낸다. H(S)는 엔트로피 함수이며, k는 클래스 개수, $P(C_i, S)$는 S의 데이터가 클래스 C_i에 속할 확률을 나타낸다.

$$IG(A,T;S) = H(S) - E(A,T;S)$$

$$E(A,T;S) = \frac{|S_1|}{|S|} H(S_1) + \frac{|S_2|}{|S|} H(S_2)$$

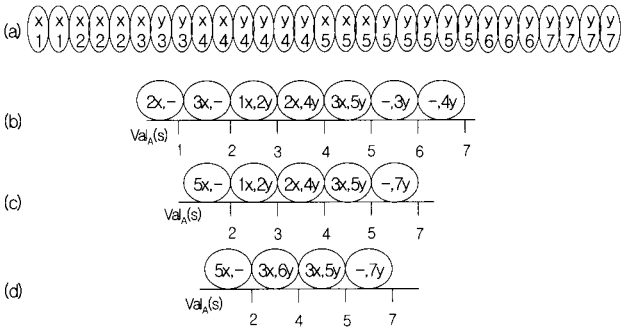
$$H(S) = -\sum_{i=1}^{k} P(C_i,S) \log P(C_i,S)$$

(그림 1) Information Gain 함수

Fayyad & Irani는 특정 수치형 변수 A에 대해서 모든 가능한 분할 포인트 중 평균 클래스 엔트로피를 가장 최소화하는 T^{min}을 분할 포인트로 선택하여 구간을 분할하고 이와 같은 방법을 두 부분 구간에 재귀적으로 적용하여 특정 조건이 만족될 때까지 구간을 이진 분할함으로써 다중 분할 포인트(Multi Split Point)를 생성하는 이진화 기법(Binarization Method)을 이용하였다[11, 12]. 특히, 이들은 평가함수의 볼록한(Convex) 특성을 이용하여 빈(Bin)과 블록(Block)을 정의함으로써 가능한 후보 분할 포인트 수를 줄이려 하

였다. 또한, Elomaa는 이러한 개념을 더 확장하여 세그먼트(Segments)라는 개념을 정의하였다[13, 14].

(그림 2)는 클래스 x, y를 갖는 데이터를 정렬한 후 값이 같은 데이터를 하나로 통합한 빈과 클래스가 같은 인접한 빈들을 통합한 블록 및 상대적 클래스 분포가 같은 서로 인접해 있는 블록을 통합한 세그먼트를 나타낸다. (그림 2)에서는 평가함수의 볼록한 특성을 이용함으로써 27개의 후보 분할 포인트에서 4개의 후보 분할 포인트로 줄일 수 있음을 나타낸다.



(그림 2) (a) 정렬된 원본 데이터; (b) 빈즈(Bins); (c) 블록(Blocks); (d) 세그먼트(Segments)

그러나 이진화 기법을 사용하는 대부분의 전역적 범주화 기법은 단일 빈즈(Single Bins)를 필요로 하기 때문에 순차적으로 트리를 생성해야하는 환경에 적합하지 못하다. 왜냐하면, 새로운 데이터가 추가될 때, 이 데이터가 반영된 범주를 생성하기 위해서는 기존의 빈즈와 병합(Merge)하여 하나의 단일 빈즈를 생성해야하기 때문이다. 특히, 이러한 기법은 데이터가 대용량이고 범주화를 수행할 변수의 범위가 매우 클 경우, 단일 빈즈를 생성하기 위해 많은 정렬 및 병합을 수행해야하며 이는 매우 비효율적이다.

2.2 결정 트리 알고리즘

대용량 데이터를 위한 대표적인 결정트리 알고리즘으로 SLIQ[3], SPRINT[4], RainForest[5] 등을 들 수 있다. 이 중 SLIQ, SPRINT는 트리의 각 노드에서 수치형 변수를 평가하기 위해 수행되는 정렬을 루트 노드에서 한번만 수행함으로써 수행 속도를 향상시켰다. 특히, SPRINT는 (그림 3)과 같은 속성 리스트(Attribute List)라는 데이터 구조를 변수별로 생성하였으며 속성 리스트의 각 레코드는 변수 값, 클래스, 레코드 인덱스(Rid)로 구성되어 있다.

SPRINT는 속성 리스트의 각 레코드의 변수 값과 클래스를 이용하여 최적의 분할 변수를 선택하였다. 분할 변수가 선택된 후 노드의 데이터 분할은 레코드 인덱스를 이용하여 수행된다. 우선, 선택된 분할 변수의 속성 리스트를 나누며 해당 레코드가 어느 서브 트리에 전달되었는지를 나타내기 위해 해당 레코드 인덱스를 해시 테이블(Hash Table)에 저장한다. 분할 변수의 레코드 인덱스가 해시 테이블에 모두 저장된 후, 비분할 변수의 속성 리스트의 분할은 해당 레코드 인덱스와 해시 테이블을 이용하여 해당 레코드가 어느 서브 트리에 전달되었는지 검색하여 수행된다.

Age	Class	Rid
17	High	1
20	High	5
23	High	0
32	Low	4
43	High	2
68	Low	3

Car	Class	Rid
Family	High	0
Sports	High	1
Sports	High	2
Family	Low	3
Truck	Low	4
Family	High	5

(그림 3) 속성 리스트(Attribute List)

SPRINT의 이러한 접근 방법은 전체 데이터를 각 변수별로 나누어 한 순간에 메모리에 적재되어야 할 데이터 양을 줄임으로써 대용량 데이터를 처리 할 수 있지만, 각 변수를 개별적으로 관리해야한다는 문제점을 안고 있으며 특히, 해시 테이블을 메모리에 적재하지 못할 경우, 해시 테이블을 나누어 분할 작업을 여러 번 수행해야 한다.

RainForest는 현재 노드에 도달한 데이터가 메모리에 적재할 수 있는 경우와 메모리에 적재할 수 없는 경우를 구분하여 적재할 수 있는 경우에는 일반적인 메모리 기반 알고리즘을 사용하고 그렇지 못한 경우는 각 변수에 대해 AVC-set(Attribute-Value Classlabel)을 생성하여 결정 트리를 생성하는 알고리즘이다. RainForest는 특정 노드의 모든 변수에 대한 AVC-set의 집합을 AVC-group이라 정의하여 사용한다. RainForest는 현재 노드에서 AVC-set 혹은 AVC-group이 메모리에 적재 되는지에 따라 다음과 같은 세 경우로 나누어 처리한다. 첫째, AVC-group 모두가 메모리에 적재되는 경우, 둘째, AVC-group은 메모리에 적재되지 않지만 각각의 AVC-set은 메모리에 적재되는 경우, 셋째 각 AVC-set 자체도 메모리에 적재되지 않는 경우이다. 메모리에 적재되지 않은 경우, 많은 입출력 오버헤드가 필요하기 때문에 메모리에 적재되는 경우보다 수행 시간이 더 필요하다.

트리를 순차적으로 생성하는 알고리즘은 새로운 데이터 추가로 인해 발생하는 분할 변수(Split Variable) 변경과 분할 포인트(Split Point) 변경 등을 처리할 수 있어야 한다. 대표적인 순차적 결정 트리 알고리즘으로 ITI[6]와 BOAT[7] 등을 들 수 있다.

ITI는 메모리 기반 순차적 알고리즘으로 분할 포인트 변경과 분할 변수 변경을 처리하기 위해 트리 재구화 기법(Tree Restructuring)을 이용한다. 트리 재구조화 기법은 단순한 포인터 연산을 통해 분할 변수 변경 등을 쉽게 할 수 있다는 장점을 가지고 있다. 그러나 ITI는 범주형 변수에 대해서는 변수가 갖는 각 값에 대한 클래스 빈도만 유지하면 되지만 수치형 변수일 경우에는, 각 값에 대한 클래스 빈도를 유지하고 값들을 정렬된 형태로 유지하고 있어야 한다. 따라서 수치형 변수의 개수가 n개이고 트리의 깊이가 d이며 특정 수치형 변수의 범위가 k일 때 n*d*k의 공간이 필요하

기 때문에 대용량 데이터 처리에 한계가 있다.

BOAT는 RainForest와 같이 데이터 메모리 적재 여부에 따라 데이터를 모두 메모리에 적재할 수 있으면 메모리 기반 알고리즘을 이용하고 그렇지 않으면 샘플링 단계(Sampling Phase)와 정리 단계(Cleanup Phase)등 두 단계를 걸쳐 트리를 생성한다. 샘플링 단계에서 전체 데이터 D로부터 샘플을 추출하고 이로부터 샘플 데이터 $D_1,...,D_b$을 추출하여 b개의 샘플 트리 $T_1,...,T_b$를 생성한 후 b개의 트리를 부트스트래핑 방식을 이용하여 최종 트리와 유사한 샘플 트리를 생성한다. 정리 단계에서는 전체 데이터를 샘플 트리에 전달한 후 샘플 트리의 분할 변수 및 분할 포인트가 최종 트리와 같은지 검증하며 최종 트리와 같으면 각 서브 트리로 검증을 계속하고 그렇지 않으면 그 노드를 잎 노드로 변환하여 트리를 다시 생성 한다. BOAT는 효율적인 분할 포인트 검증을 위해 부트스트래핑에서 생성된 신뢰구간을 이용하며 최종 분할 포인트가 신뢰구간 안에 있으면 그 구간에 존재하는 데이터를 서브 트리에 전달하고 신뢰구간 밖에 있으면 그 노드에서 트리를 다시 생성한다. 이러한 방식으로 BOAT는 기존 트리를 샘플 단계에서 생성된 샘플 트리로 간주함으로써 쉽게 순차적 알고리즘으로 확장될 수 있다.

BOAT는 수치형 분할 변수에 대해 신뢰구간과 이 구간에 포함되는 데이터를 유지하고 있기 때문에 신뢰구간 내에서의 분할 포인트 변경은 효율적으로 처리 할 수 있다. 그러나 분할 변수와 최종 분할 변수가 다른 경우와 최종 분할 포인트가 신뢰구간을 넘어서는 경우에는 그 노드에서 트리를 다시 생성해야한다. 만약, 트리의 루트 노드의 분할 변수가 변경된다면 트리를 처음부터 다시 생성해야한다.

이러한 대부분의 결정 트리 알고리즘은 수치형 변수 처리를 위해 지역적 범주화 기법에 기반을 두고 있다. 그러나 범주화는 정렬된 형태의 수치형 데이터를 요구하기 때문에 대용량 데이터를 처리해야하는 상황에서 전체 데이터에 대해 한번만 정렬을 수행하는 전역적 범주화 기법이 모든 노드에서 매번 정렬을 수행하는 지역적 범주화보다 적합하다. 본 논문에서는 이를 위해 다중 빈즈를 이용한 전역적 순차적 범주화와 이를 이용한 순차적 결정 트리 알고리즘을 제안한다.

3. 다중 빈즈를 이용한 순차적 범주화

본 논문에서는 전역적 범주화에서 두 샘플 분할 포인트 v_i, v_{i+1}로 정의되는 샘플 포인트 구간 $[v_i, v_{i+1}]$에 대해 평균 클래스 엔트로피 하위 경계값을 예측함으로써 다중 분할 포인트(Multi Split Point)를 생성하는 기법을 이용한다. 또한 단일 빈즈(Single Bins)에서 범주화를 수행하는 일반적인 전역적 범주화 기법과 달리 다중 빈즈(Multi Bins)에서 범주화를 수행할 수 있도록 확장함으로써 순차적인 범주화도 가능하도록 한다[15].

3.1 샘플 포인트를 이용한 전역적 범주화

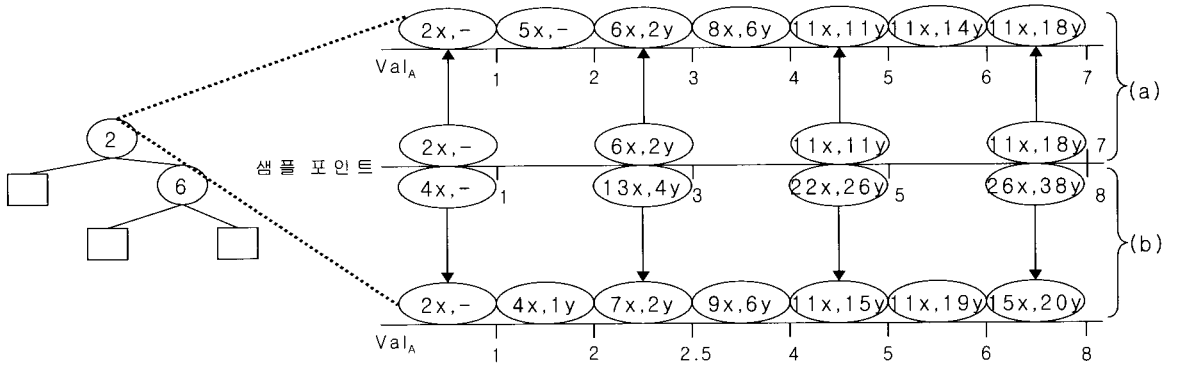
일반적으로 전역적 범주화는 (그림 2)와 같은 빈즈을 이용하여 수행된다. 최적의 분할 포인트를 선택하기 위해 왼쪽에서 오른쪽으로 빈즈를 읽어 가면서 각 분할 포인트에 대해 평균 클래스 엔트로피를 계산한다. 그러나 임의의 위치에 있는 샘플 분할 포인트를 추출하고 이를 평가하기 위해서는 (그림 2)와 같은 빈즈를 이용하는 것은 비효율적이다. 왜냐하면 특정 샘플 분할 포인트 v의 평균 클래스 엔트로피를 계산하기 위해서는 v이전의 빈즈를 읽어야 하기 때문이다. 본 논문에서는 이러한 문제를 해결하기 위해 누적된 빈즈를 이용한다. 누적된 빈즈에서는 클래스 개수가 k일 때, 변수 X의 임의의 분할 포인트 v의 통계치 벡터는 $(n^v_1,...,n^v_k)$이며 $n^v_i = |\{t : t.X \le v \wedge t.C=i\}|$ 이다. (그림 2) (b)를 누적된 빈즈로 변환하면 값 2에 대한 누적 통계치 벡터는 (5x, 0y)이고 값 4에 대한 통계치 벡터는 (8x, 6y)가 된다. 누적된 빈즈에서 임의의 분할 포인트 v의 평균 클래스 엔트로피 계산은 v의 빈에 이미 필요한 클래스 분포가 누적되어 있기 때문에 이전 빈들을 읽을 필요 없이 v의 빈만을 이용하여 계산할 수 있다.

누적된 빈즈를 생성한 후 최적의 분할 포인트를 선택하는 방법은 다음과 같다. 구간 $[v_1, v_z]$ 중에서 임의의 샘플 분할 포인트 q개를 추출한다. 추출된 샘플 포인트에 평균 클래스 엔트로피를 적용한 후 최소의 엔트로피를 Ent^{min}라 한다. 이 경우 Ent^{min}이 q개의 샘플 포인트에서뿐만 아니라 모든 샘플 포인트 구간$[v_i, v_{i+1}]$에서도 최소임을 증명해야하며 이를 위해 정리 1[7]을 이용한다.

정리 1. 특정 수치형 변수 X의 값 v 및 k개의 값을 갖는 클래스 변수 C에 대해, $t.X$는 데이터 t의 변수 X의 값일 때, v의 클래스 i에 대한 통계치 n^v_i를 $n^v_i = |\{t : t.X \le v \wedge t.C=i\}|$이라 하며, $v_1 < v_2$를 만족하는 X의 값 v_1, v_2에 대해 k개의 클래스에 대한 통계치 벡터를 각각 $(n^{v1}_1,...,n^{v1}_k)$, $(n^{v2}_1,...,n^{v2}_k)$라 한다. $P^{v1,v2}$를 v_1과 v_2사이에 발생할 수 있는 모든 통계치 벡터들의 집합이라 하고 S를 $(n^{v1}_1,...,n^{v1}_k)$, $(n^{v2}_1,...,n^{v2}_k)$로 유추할 수 있는 2^k개의 가상 통계치 벡터라 정의한다($S = \{(n^{v1}_1,...,n^{v1}_k), (n^{v2}_1, n^{v1}_2...,n^{v1}_k),...,(n^{v2}_1,...,n^{v2}_{k-1}, n^{v1}_k),(n^{v2}_1,...,n^{v2}_k)\}$). imp를 평가 함수라 하면 S와 $P^{v1,v2}$는 다음을 만족한다.

$$\min_{(n_1,...,n_k)\in S} imp\,(n_1,..,n_k) \le \min_{(n_1,...,n_k)\in P^{x1,x2}} imp\,(n_1,..,n_k)$$

각 샘플 포인트 구간 $[v_i, v_{i+1}]$에서 가상의 통계치 벡터 S를 추출하여 이로부터 하위 경계값, Ent^{est}를 계산하여 $Ent^{est} \ge Ent^{min}$이면 다음 구간 $[v_{i+1}, v_{i+2}]$로 넘어간다. 그렇지 않으며, $[v_i, v_{i+1}]$의 중앙값 v_{mid}을 추출하여 구간$[v_i, v_{i+1}]$을 두 구간 $[v_i, v_{mid}]$, $[v_{mid}, v_{i+1}]$로 분할한다. 검증은 $[v_i, v_{mid}]$에 대



(그림 4) 범주 트리와 노드 정보

해 계속적으로 이루어지며 샘플 포인트 구간을 더 이상 분할할 수 없는 오직 하나의 값만 가질 때까지 이루어진다.

3.2 다중 빈즈를 이용한 순차적 범주화

대부분의 범주화 알고리즘은 단일 빈즈(Single Bins)를 필요로 하기 때문에 새로운 데이터가 추가될 경우, 이를 반영한 범주를 생성하기 위해 기존의 빈즈와 병합한 후 처음부터 범주화를 수행한다. 이러한 방법은 데이터가 소용량일 경우 메모리에서 효율적으로 수행될 수 있지만 데이터가 대용량이고 범주화를 수행할 변수의 범위가 매우 크면 매우 비효율적인 방법이다.

본 논문에서는 이러한 문제점을 해결하기 위해 트리 형태의 범주 모델을 유지하고, 새로운 빈즈가 추가되면 기존의 빈즈와 병합하지 않고 이를 범주 모델에 반영하여 순차적으로 범주를 생성하는 기법을 이용한다[15]. (그림 4)는 최종 분할 포인트가 2, 6인 범주 트리를 나타내며 루트 노드에는 최종 분할 포인트 생성에 이용된 샘플 분할 포인트들이 유지된다. (그림 4) (a)는 전체 구간 [1, 7]에서 샘플 분할 포인트를 추출한 상태를 나타내며 각 샘플 포인트는 해당 빈에 대한 누적된 통계치 벡터를 가지고 있다. 새로운 빈즈가 추가되면 각 샘플 포인트는 (그림 4) (b)와 같이 새로 추가된 빈즈를 포함하도록 갱신된다. 샘플 포인트 갱신은 각 샘플 포인트 v의 통계치 벡터 $(n^{v}_{1},..,n^{v}_{k})$가 $n^{v}_{i} = |\{t : t.X \le v \wedge t.C=i\}|$의 조건을 만족하도록 샘플 포인트 값과 같거나 작으면서 가장 큰 빈의 클래스 분포를 누적시킨다.

i번째 순차적 범주화는 i-1번째 범주화에서 추출된 샘플 분할 포인트에 3.1절과 같이 평균 클래스 엔트로피 함수를 적용함으로써 시작하며 이들 중 최소의 엔트로피를 Ent^{min}라 한다. Ent^{min}이 추출된 샘플 포인트에서뿐만 아니라 모든 샘플 포인트 구간$[v_i, v_{i+1}]$에서도 최소임을 증명해야하며 이를 위해 정리 1을 이용한다. 그러나 구간에 대한 평균 클래스 엔트로피의 하위 경계값 예측을 이용하는 위 방법에서는 실제 추출한 샘플 분할 포인트의 평균 클래스 엔트로피 보다 구간에 대한 평균 클래스 엔트로피를 낮게 평가하는 과소평가(Underestimate) 현상이 발생할 수 있다. 과소평가 현상이 발생하면, 실제 존재하지 않는 분할 포인트를 찾기 위해 구간에 대한 검색을 불필요하게 계속 수행해야하며 이로 인해 메모리에서 유지해야하는 샘플 포인트의 수도 증가한다. 본 논문에서는 이러한 과소평가 현상을 줄이기 위해 i번째 범주화에서는 i-1번째 범주화에서 발생한 평가 오차를 이용하여 구간에 대한 예측치를 보정하는 방법을 사용하였다. 범주화를 수행하는 구간 $[v_s, v_e]$에 대해, 예측 구간 $[v_i, v_{i+1}]$ 및 $[v_i, v_{i+1}]$의 중간 샘플 포인트가 v_{mid}이고 v_{mid}의 통계치 벡터가 $(n^{vmid}_{1},...,n^{vmid}_{k})$이면 평균 오차는 (그림 5)와 같다.

$$\text{평균오차}(E) = \frac{\sum \text{imp}(n^{vmid}_{i},\ldots,n^{vmid}_{k}) - \min_{\{n_1,\ldots,n_k\}\in S}(\text{imp}(n_i,\ldots,n_k)) \times \frac{v_{i+1}-v_i}{v_e-v_s}}{N}$$

$\text{imp}(n^{vmid}_{i},\ldots,n^{vmid}_{k})$: v_{mid}의 평균 클래스 엔트로피

$\min_{\{n_1,\ldots,n_k\}\in S}(\text{imp}(n_i,\ldots,n_k))$: 구간에 대한 예측치

$\frac{v_{i+1}-v_i}{v_e-v_s}$: 현재 예측 구간의 비율

N : 오차 발생 횟수

(그림 5) 평균 오차

(그림 5)에서 실제 평균 엔트로피와 구간에 대한 예측치 차에 구간의 비율을 가중한 이유는 예측 구간이 클 수 록 오차가 증가하기 때문에 이를 정규화하기 위해서이다. i번째 범주화에서 구간에 대한 예측은 i-1번째 범주화에서의 평균 오차를 이용하여 보정한다. i번째 범주화에서 범주화를 수행하는 구간 $[v_s, v_e]$에 대해, 구간 $[v_i, v_{i+1}]$의 보정된 예측치는 (그림 6)과 같다. 보정된 예측치는 예측치에 평균오차에 구간의 비율만큼 더한 값이다. α는 0과 1사이의 값을 갖으며 α = 0일 경우, 예측치를 보정하지 않으며 α = 1일 경우, 평균 오차만큼 보정하는 결과가 된다.

$$\text{보정된 예측치} = \min_{\{n_1,\ldots,n_k\}\in S}(\text{imp}(n_i,\ldots,n_k)) + E \times \frac{v_{i+1}-v_i}{v_e-v_s} \times \alpha \quad (0 \le \alpha \le 1)$$

(그림 6) 보정된 예측치

4. 전역적 범주화를 이용한 트리의 순차적 생성

전처리로서 전역적 범주화를 이용하여 생성된 트리에서 새로운 데이터가 추가될 경우, 이를 반영한 트리를 순차적

으로 생성하기 위해서는 첫째, 이 새로운 데이터를 반영한 범주를 재생성해야 하고, 둘째, 기존의 범주와 새로 생성된 범주를 비교하여 새로운 범주에 맞게 트리를 재생성해야 하며, 셋째, 새로운 데이터 추가로 인해 발생하는 분할 변수 변경을 처리해야한다.

본 논문에서는 범주 재생성을 위해 3장에서 설명한 기법을 이용하였다. 두 번째의 범주 변화 처리를 위한 가장 간단한 방법은 범주 변화가 발생한 노드에서 트리를 다시 생성하는 것이다. 그러나 이 방법은 트리의 루트 노드에서 범주의 변화가 발생할 경우, 트리를 처음부터 다시 생성해야 한다. 본 논문에서는 효율적인 범주 변화 처리를 위해 신뢰구간과 트리 재구조화 기법을 이용하는 방법을 제안하며 본 장에서 설명한다. 세 번째의 분할 변수 변경은 기존 순차적 기법의 다양한 기법을 적용할 수 있으며 ITI는 트리 재구조화 기법을 이용하며 BOAT는 변경노드에서 트리를 다시 생성한다.

4.1 신뢰구간

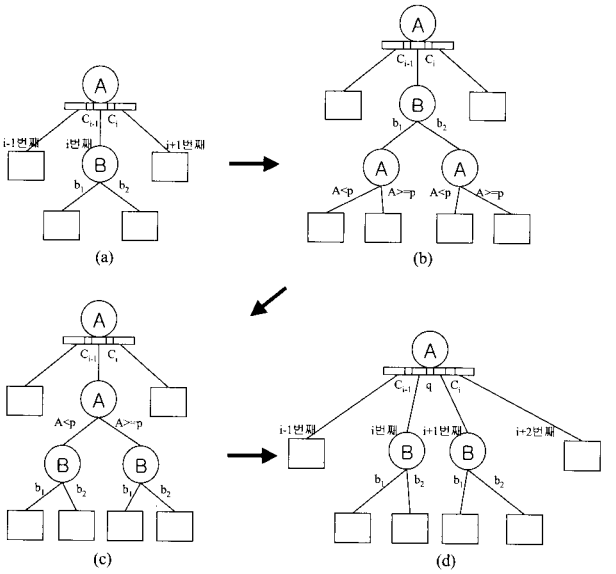
신뢰구간은 최종 분할 포인트가 발생할 가능성이 많은 구간으로 순차적 트리생성에서 효율적으로 분할 포인트 변경을 처리하기 위해 이용된다. 지역적 범주화를 이용하는 BOAT에서는 부트스트래핑을 이용하여 이 구간을 생성하지만 본 논문에는 이와 달리 전역적 범주화 후 샘플 포인트들의 흩어진 정도(Sparseness)를 이용하여 이 신뢰구간을 생성한다. 범주화 후 추출된 샘플 분할 포인트들의 분포를 보면 최종 분할 포인트 주위에 많은 샘플 포인트들이 존재하고 최종 분할 포인트에서 멀어질수록 샘플 분할 포인트의 수가 적음을 알 수 있다. 즉, 최종 분할 포인트 주위에서는 샘플 포인트의 흩어진 정도가 낮고 최종 분할 포인트에서 멀어질수록 샘플 포인트의 흩어진 정도가 높은 현상을 발견할 수 있다. 이는 최종 분할 포인트 주위의 샘플 포인트들의 평가치가 최종 분할 포인트의 평가치와 유사하여 더 좋은 분할 포인트를 검색하려는 시도가 많았고 이로 인해 많은 샘플 포인트들을 추출했기 때문이다. 따라서 최종 분할 포인트 주위에서 흩어진 정도가 낮은 구간은 다음 범주화에서 최종 분할 포인트가 존재할 가능성이 높으며 이를 신뢰구간으로 지정하였다. 인접한 샘플 포인트 v_i, v_{i+1}의 흩어진 정도는 인접 샘플 포인트의 차 $v_{i+1} - v_i$ 로 정의된다. 신뢰구간의 왼쪽 경계는 최종 분할 포인트로부터 왼쪽으로 이동하며 인접 분할 포인트의 흩어진 정도가 임계값(threshold)보다 큰 첫 번째 샘플 포인트로 지정하며 오른쪽 경계는 오른쪽으로 이동하며 임계값보다 큰 첫 번째 샘플 포인트로 지정한다. i번째 최종 분할 포인트의 신뢰구간 C_i를 $[l_i, u_i]$로 표기하며 실험에서는 흩어진 정도의 평균을 임계값으로 사용하였다.

4.2 초기 트리 생성

대용량 데이터를 위한 결정 트리 알고리즘으로 SPRINT, RainForest, BOAT 등이 있으며 초기 트리 생성을 위해 이러한 알고리즘을 이용할 수 있다. 본 논문에서는 그 중 BOAT를 트리 생성 알고리즘으로 선택하였으며 그 이유는 BOAT가 신뢰구간 및 순차적 트리 생성을 지원하기 때문이다.

전역적 범주화 기반 BOAT의 트리 생성은 지역적 범주화를 이용한 경우에서와 같이 샘플링 단계와 정리 단계 등 두 단계를 걸쳐 이루어진다. 그러나 지역적 범주화 기반 BOAT와 다른 점은 지역적 범주화 기반은 수치형 분할 노드에 하나의 분할 포인트를 결정하기 위해 하나의 신뢰구간을 유지하는 반면, 전역적 범주화를 사용할 경우, 수치형 변수에 다중 분할 포인트를 생성하기 여러 개의 신뢰구간을 생성한다. 즉, 수치형 분할 노드 X가 k개의 분할 포인트를 갖는다면 이 노드에는 범주화 과정에서 생성된 k개의 신뢰구간 및 k개의 신뢰구간을 만족하는 데이터 집합 F^i_X, k+1개의 서브 트리가 존재한다. i번째 신뢰구간 C_i이 $[l_i, u_i]$이면 i번째 분할 포인트 p_i는 $l_i <= p_i <= u_i$ 이며 노드 X에 전달된 데이터 F_X에 대해 i번째 신뢰구간을 만족하는 데이터 집합 F^i_X는 $\{t \in F_x : l_i <= t.X <= u_i\}$ 이다. F^i_X를 유지하는 이유는 다음 순차적 범주화에서 생성된 분할 포인트 p_i'가 $l_i <= p_i' <= u_i$이면 F^i_X를 p_i'로의 분할만으로 이 변화를 처리할 수 있기 때문이다. 트리 생성 후, 트리 생성에 이용된 데이터는 다음 번 순차적 트리 생성을 위해 디스크에 저장되며 신뢰구간에 만족하는 데이터는 해당 수치형 분할 노드에 관리되고 나머지 데이터는 잎 노드에서 관리된다.

4.3 범주 변화 처리 알고리즘



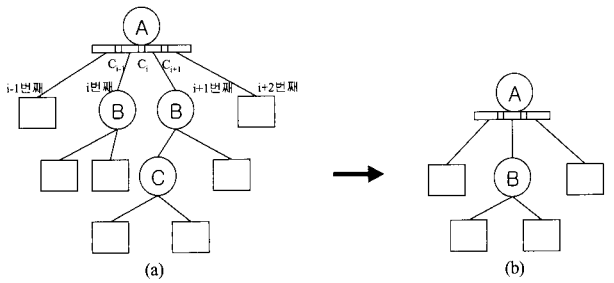
(그림 7) 신뢰구간 C_i, C_{i+1} 사이에 발생한 분할 포인트 처리 과정

전역적 범주화를 이용하여 생성된 트리에서 새로운 데이터가 추가될 경우, 이를 반영한 트리를 생성하기 위

해서는 범주 변화를 처리해야한다. 범주 변화는 기존의 범주와 새로 생성된 범주를 비교함으로써 알 수 있다.

신뢰구간을 유지하고 있는 수치형 분할 노드에는 첫 번째, 노드에서 유지하고 있는 신뢰구간 C_{i-1}과 C_i사이에 새로운 분할 포인트가 추가되는 경우, 두 번째, i번째 신뢰구간 C_i에 분할 포인트가 생성되지 않은 경우, 세 번째, 신뢰구간에 두개 이상의 분할 포인트가 발생한 경우 네 번째, 신뢰구간에 오직 하나의 분할 포인트가 존재하는 경우 등 네 가지 형태의 범주 변화가 발생할 수 있다.

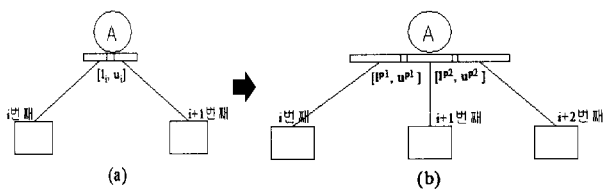
(그림 7)은 범주 변화 첫 번째 경우로서, 수치형 분할 변수 A의 신뢰구간 C_{i-1}과 C_i사이에 새로운 분할 포인트가 추가된 경우의 처리과정을 나타낸다. A는 수치형 변수이며 C_i는 변수 A를 범주화하는 과정에서 생성된 i번째 신뢰구간이며 구간 $[l_i, u_i]$를 갖는다. B는 범주형 변수이며 b1, b2 등 두개의 값을 갖는다. 그림에서 원은 분할 노드를 나타내며 사각형은 잎 노드를 나타낸다. (그림 7) (a)는 새로운 데이터 추가 이전의 결정 트리를 나타내며 노드 A의 i번째 서브 트리는 노드 A에 전달된 데이터 F_A중 $\{t \in F_A : u_{i-1} < t.A < l_i\}$인 데이터로 구성되었다. 이 트리에 새로운 데이터가 추가되어, 변수 A의 범주에 $u_{i-1} < p < l_i$ 을 만족하는 새로운 분할 포인트 p가 신뢰구간 C_{i-1}과 C_i번째 사이에 생성되었다. 이는 i-1과 i+1번째 서브 트리 사이에 하나의 서브 트리가 더 추가되어야 함을 의미하며 이는 i번째 서브 트리를 구성한 데이터 $\{t \in F_A : u_{i-1} < t.A < l_i\}$를 추가된 분할 포인트 p로 더 분할함으로써 처리할 수 있다. (그림 7) (b)는 A의 i번째 서브 트리를 분할하기 위해 i번째 서브 트리의 잎 노드인 B의 서브 트리를 변수 A의 새로운 분할 포인트 p로 분할한 것을 나타낸다. (그림 7) (c)는 분할 포인트 p를 분할 노드 A에 포함시키기 위해 노드 A를 원래 위치인 노드 B의 상위 노드로 끌어올리는 과정을 나타내며 이는 트리 재구조화 기법을 이용하여 수행된다. (그림 7) (d)는 i번째와 i+1번째 서브 트리로 B를 연결함으로써 분할 노드 A에 새로운 분할 포인트 p를 포함시킬 수 있음을 나타내며 이를 통해 기존의 $\{t \in F_A : u_{i-1} < t.A < l_i\}$로 구성된 트리를 $\{t \in F_A : u_{i-1} < t.A < p\}$와 $\{t \in F_A : p <= t.A < l_i\}$로 구성된 두개의 서브트리로 분할하였다.



(그림 8) 신뢰구간에 분할 포인트가 발생하지 않을 경우 처리과정

(그림 8)은 범주 변화 두 번째 경우로서, i번째 신뢰구간 C_i에 분할 포인트가 발생하지 않은 경우 처리 과정을 나타낸다. (그림 8) (a)는 새로운 데이터 추가 이전의 결정 트리를 나타내며, 분할 노드 A는 신뢰구간 C_{i-1}, C_i, C_{i+1} 등을 갖으며 이로부터 생성된 i-1번째, i번째, i+1번째, i+2번째 서브 트리를 갖고 있다. 이 중 i번째 서브 트리는 $\{t \in F_A : u_{i-1} < t.A < l_i\}$인 데이터로 구성되며 i+1번째 서브 트리는 $\{t \in F_A : u_i < t.A < l_{i+1}\}$인 데이터로 구성되어 있다. 새로운 데이터 추가로 인해 신뢰구간 C_i에 분할 포인트가 생성되지 않으면 i번째 서브 트리와 i+1번째 서브 트리의 경계인 신뢰구간 C_i가 없어지므로 두 서브 트리를 병합하여 하나의 서브 트리로 생성해야한다. 두개의 서브 트리 병합은 부트스트래핑을 이용하여 이루어진다. (그림 8) (b)는 (그림 8) (a)에서 분할 변수가 B인 두 서브 트리를 부트스트랩핑을 이용하여 병합한 결과를 나타낸다. 부트스트래핑은 두 노드를 비교하여 분할 변수가 같으면 두 노드를 통합하고 그렇지 않으면 잎 노드로 변환한다[7].

(그림 9)는 범주 변화 세 번째 경우로서, 신뢰구간 C_i에 두개 이상의 분할 포인트가 발생한 경우의 처리과정을 나타낸다. (그림 9) (a)는 구간 $[l_i, u_i]$인 i번째 신뢰구간 C_i를 갖는 분할 노드 A를 나타내며 i번째 서브 트리는 $\{t \in F_A : u_{i-1} < t.A < l_i\}$인 데이터로 구성되며 i+1번째 서브 트리는 $\{t \in F_A : u_i < t.A < l_{i+1}\}$인 데이터로 구성되어 있다. 새로운 데이터가 추가되어 이 신뢰구간에 신뢰구간이 각각 $[l^{p1}, u^{p1}]$, $[l^{p2}, u^{p2}]$인 새로운 분할 포인트 p_1, p_2가 생성될 경우, 노드 A에서 유지하고 있는 F^i_A를 분할하여 서브 트리를 확장해야한다. (그림 9) (b)는 F^i_A를 분할 포인트 p_1, p_2로 분할하여 서브 트리를 더 생성한 결과를 나타낸다. (그림 9) (b)에서 i번째 서브 트리는 $\{t \in F_A : u_{i-1} < t.A < l^{p1}\}$인 데이터로 구성되고 i+1번째 서브 트리는 $\{t \in F_A : u^{p1} < t.A < l^{p2}\}$인 데이터로 구성되며 i+2번째 서브 트리는 $\{t \in F_A : u^{p2} < t.A < l_{i+1}\}$인 데이터로 구성된다.



(그림 9) 신뢰구간에 하나 이상의 분할 포인트가 생성될 경우 처리과정

발생할 수 있는 범주 변화 네 번째 경우는 신뢰구간에 오직 하나의 분할 포인트가 존재하는 경우이며, 이는 기존의 분할 포인트와 새로운 분할 포인트가 같거나 신뢰구간 내에서 약간의 변경이 발생한 경우이다. 이 경우에는 노드 A에서 유지하고 있는 $\{t \in F_A : l_i < t.A < u_i\}$를 만족하는 데이터인 F^i_A를 새로운 분할 포인트 p를 이용

하여 서브 트리로 분할하면 된다. 이는 최종 분할 포인트가 결정되기까지 서브 트리로의 전달이 유보된 데이터를 최종 분할 포인트가 결정되면서 서브 트리에 전달하는 것을 의미한다.

(그림 10)은 새로운 범주를 트리에 적용하는 알고리즘을 나타낸다. step2(a)는 두 신뢰구간 사이에 새로운 분할 포인트가 생성된 경우의 처리 과정을 나타내고, step2(b), step2(c)는 하나의 신뢰구간 내에서 발생하는 분할 포인트 변경에 대한 처리 과정을 나타낸다. step2(a)에서 신뢰구간 사이에 분할 포인트가 존재하는지 검사하여 존재하면 split_variables에 해당 변수를 추가한 후 해당 서브트리에 대해 재귀적으로 adjust_newDiscretization를 호출한다. 잎 노드에 도착하면 루트 노드에서 잎 노드까지의 패스(Path)에서 신뢰구간 사이에 분할 포인트가 추가된 분할 노드를 이용하여 노드 분할을 수행한다(step1). 이는 루트 노드에서 잎 노드까지 패스에서 변경이 발생한 분할 노드 개수에 관계없이 오직 한번만 잎 노드의 데이터를 읽어 분할 포인트 추가를 처리하기 위해서이다.

잎 노드의 데이터 수가 N_1이고 신뢰구간 만족 데이터 수가 N_2라면 step2(a) 와 step2(b), step2(c)는 각각 $O(N_1)$과 $O(N_2)$의 시간이 걸리며 일반적으로 $N_1>N_2$이므로 step2(a)가 step2(b)와 step2(c)보다 시간이 더 많이 걸린다. 또한, 이러한 step2의 데이터 분할은 디스크에 저장된 데이터의 메모리 적재를 필요로 하기 때문에 이는 많은 메모리를 요구하며 이후 트리 생성에 디스크 I/O를 발생시킨다.

adjust_newDiscretization(node, split_variables)
step1. node가 잎 노드이면 split_variables를 이용하여 node 분할
step2. node가 범주 변화가 있는 수치형 변수 X의 분할노드이면
  step2(a) 모든 신뢰구간 사이 C_{i-1}과 C_i 사이 분할 포인트 존재 검사
    step2(a1) C_{i-1}과 C_i 사이에 새로운 분할 포인트가 존재하면 split_variables에 X 추가
    step2(a2) adjust_newDiscretization(node->sub_i, split_variables)
    step2(a3) sub_i의 분할 노드가 X가 되도록 트리 재구조화
    step2(a4) 분할 노드 X의 서브 트리 재정리
  step2(b) 모든 신뢰구간 C_i에 대해 C_i에 존재하는 분할 포인트 개수 검사
    step2(b1) 분할 포인트 개수가 0이면 sub_i와 sub_{i+1} 통합
    step2(b2) 분할 포인트 개수가 2이상이면 서브 트리 생성
  step2(c) F^i_X에 존재하는 데이터 서브 트리에 전달
step3. node가 범주형 변수의 분할 노드이거나 변화가 없는 수치형 분할 노드이면
  step3(a) 모든 서브 트리 sub_i에 대해 adjust_newDiscretization(node->sub_i, split_variables)

(그림 10) 순차적 트리 생성을 위한 범주 변화 처리 알고리즘

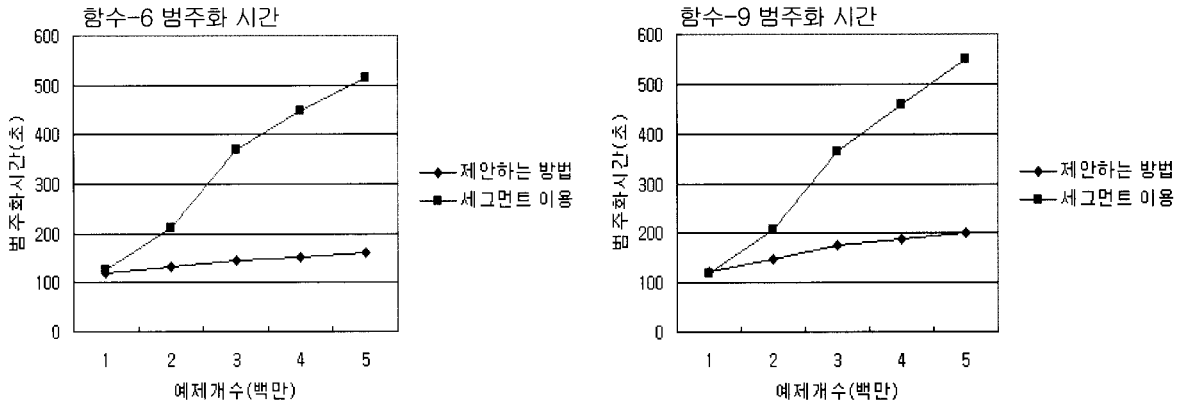
5. 실험 및 성능 평가

트리 생성 알고리즘에 관한 성능 평가로서 분류의 정확도와 트리를 생성하는 속도를 가장 많이 사용한다. 본 논문에서 제안하는 접근 방법은 Information Gain 및 Gain Ratio와 같은 평가 함수를 사용하기 때문에 생성되는 트리는 평가함수가 같다면 전역적 범주화 기법을 이용한 메모리 기반 알고리즘이 생성하는 트리와 구조가 같다. 따라서 분류의 정확도는 결정 트리 및 Naive-Bayes에서 지역적 범주화와 전역적 범주화의 정확도를 비교 실험한 [8]과 같다. 따라서 본 논문에서는 트리를 생성하는 속도에 대해서만 비교 실험하였으며, 모든 알고리즘은 JAVA를 이용하여 구현하여 Pentium-III 1G, 메모리 128MB, WINDOW 2000 환경 하에서 이루어졌다. 본 논문에서는 대용량 데이터에 대한 제안하는 접근 방법의 성능을 평가하기 위해 [16]에서 제안한 가상의 데이터를 이용하였다. 이 가상의 데이터는 9개의 변수와 한 개의 클래스 변수로 구성되어 있으며

〈표 1〉 분류함수-6, 9와 변수 설명

분류함수-6 Group A:
((age<40)∧(50K≤salary+commission≤100K)) ∨
((40≤age<60)∧(75K≤salary+commission≤125K))∨
((age≥60) ∧(25K≤salary+commission ≤75K))
분류함수-9 Group A:
disposable > 0
where disposable = (0.67×(salary+commission) -5000×elevel - (0.2×loan - 10000))

변수	값
salary	20000에서 150000까지 균등하게 분포
commission	salary≥75000⇒commission=0 그렇지 않으면 10000에서 75000까지 균등하게 분포
age	20에서 80까지 균등하게 분포
elevel	0부터 4까지 균등하게분포
car	1부터 20까지 균등하게 분포
zipcode	1부터 9까지 균등하게 분포
hvalue	0.5k100000에서 1.5k100000까지 균등하게 분포 (k는 zipcode 값)
hyears	1부터 30까지 균등하게 분포
loan	0부터 500000까지 균등하게 분포



(그림 11) 함수-6, 함수-9에 대한 순차적 범주화 시간

10개의 분류 함수로 다양한 복잡도의 데이터를 생성할 수 있다. 본 실험에서는 <표 1>과 같은 분류 함수 6, 9를 사용하였다. 분류 함수-6에서는 세 개의 변수가 데이터 분류에 결정적인 역할을 수행하며 분류 함수-9에서는 네 개의 변수의 선형적 조합이 데이터의 클래스를 결정한다. 함수 6, 9를 선택한 이유는 복잡도에 따른 알고리즘의 성능을 평가를 할 수 있으며 이들 함수가 10개의 함수 중에서 실제 데이터와 유사한 복잡도를 가지고 있기 때문이다.

첫 번째 실험으로 대용량 데이터에 대해 순차적인 범주 재생성을 위해 본 논문에서 제안하는 다중 빈즈에서 범주화를 수행하는 접근 방법의 성능을 평가하였다. 본 논문에서는 (그림 11)과 같이 데이터의 개수를 1백만에서 5백만까지 데이터를 1백만씩 증가시키면서 실험하였으며 Elomma의 세그먼트를 이용한 접근 방법과 비교하였다. 범주화 시간에는 다음번의 순차적 범주화를 위한 빈즈를 디스크에 저장하는 시간도 포함되어 있다.

(그림 11)에서 본 논문에서 제안하는 접근 방법이 더 좋은 성능을 보였는데 그 이유는 새로 추가된 데이터가 반영된 범주를 생성하기 위해 기존의 방법은 유지하고 있는 빈즈와 병합을 수행해야하지만 본 논문에서 제안하는 방법은 기존 빈즈와 병합 없이 유지하고 있는 샘플 포인트의 인덱스와 클래스 분포 정보만 갱신함으로서 새로 추가된 데이터가 반영된 범주를 생성할 수 있기 때문이다. 특히, 세그먼트를 이용한 실험에서 실수형 변수 hvalue는 메모리 상에서 병합될 수 없어 디스크에서 병합을 수행하였고 디스크에서 범주화를 수행했기 때문에 더 많은 시간이 걸렸다.

두 번째 실험은 i-1번째 범주화에서 발생한 평균오차를 이용하여 i번째 범주화에서 구간 예측을 보정할 경우, 추출된 샘플수와 생성된 분할 포인트의 일치도에 대해 실험하였다. 본 실험은 1백만에서 5백만까지 1백만 데이터씩 증가시키면서 α가 0에서 0.7일 경우에 대해 실험하였다.

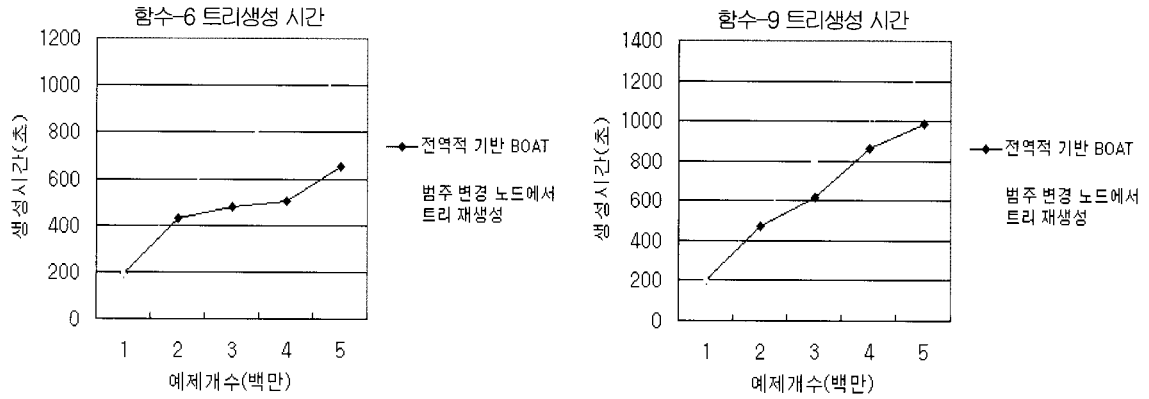
<표 2>는 함수 6, 9에서 구간에 대한 예측을 보정한 결과를 나타낸다. 포인트 수는 수치형 변수 salary, commission, age, hvalue, hyear, loan 에 대해 범주화를 수행하는 과정에서 추출된 전체 샘플 포인트 수를 나타내며 일치도는 보정치를 이용하여 생성한 최종 분할 포인트와 α=0일 경우 생성된 최종 분할 포인트와의 일치하는 비율을 나타낸다. <표 2>를 보면 함수-9는 함수-6에 비해 α=0.3 이후 일치도가 많이 떨어졌는데 이는 최종 분할 포인트의 평가치와 그 주변의 샘플 포인트의 평가치가 유사하여 보정치를 이용할 경우 최종 분할 포인트가 존재하는 구간을 검증하는 않은 경우가 많았기 때문이다. 함수-6은 α<=0.5일 때, 함수-9의 경우 α<=0.3일 때 일치도가 90%이상이며 이 시점

〈표 2〉 (a) 함수-6에 대한 보정 결과

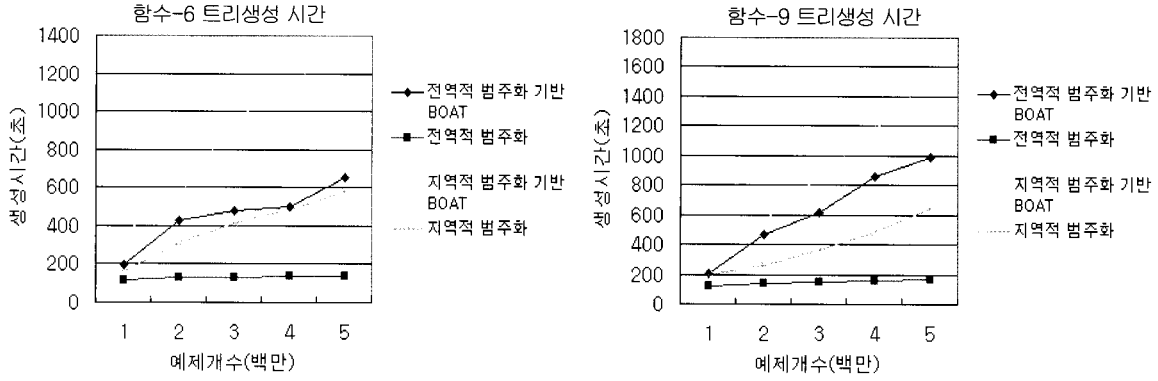
	α = 0		α = 0.4		α = 0.5		α = 0.6		α = 0.7	
	포인트수	일치도	포인트수	일치도	포인트수	일치도	포인트수	일치도	포인트수	일치도
1	15035	100	15035	100	15035	100	15035	100	15035	100
2	22558	100	19132	100	19075	94.11	19026	82.35	18815	73.52
3	28007	100	21543	100	21095	97.29	21051	94.59	20712	83.78
4	33409	100	23433	100	22287	100	22233	97.29	21658	81.08
5	39036	100	26188	100	24898	85	24891	82.5	24118	77.5

〈표 2〉 (b) 함수-9에 대한 보정 결과

	α = 0		α = 0.2		α = 0.3		α = 0.4		α = 0.5	
	포인트수	일치도	포인트수	일치도	포인트수	일치도	포인트수	일치도	포인트수	일치도
1	31412	100	31412	100	31412	100	31412	100	31412	100
2	47960	100	42144	96.55	39966	94.25	38337	85.05	37031	80.45
3	63844	100	53732	96.11	49794	94.17	46551	88.34	44029	82.52
4	79932	100	66007	92.30	60555	91.45	56181	82.05	52747	74.35
5	96076	100	77144	98.43	69529	90.62	64465	75.78	60206	66.40



(그림 12) 함수-6, 함수-9에 대한 범주 변화 처리 알고리즘 비교



(그림 13) 함수-6, 함수-9에 대한 순차적 트리생성 속도

에 추출된 샘플 포인트 수는 α=0인 경우와 비교하여 함수-6은 3천에서 1만 5천 샘플 포인트 정도를 줄일 수 있었고 함수-9의 경우 8천에서 3만 샘플 포인트를 줄일 수 있었다. 이는 평균 오차를 이용한 보정치를 사용할 경우, 생성된 최종 분할 포인트의 정확도를 유지하면서 메모리에 유지해야하는 샘플 포인트 수를 줄일 수 있기 때문에 대용량 데이터를 처리해야하는 상황에 보다 효율적인 방법임을 의미한다.

세 번째 실험은 범주 변화가 발생하는 경우 이를 처리하기 위해 본 논문에서 제안하는 범주 변화 처리 알고리즘의 성능을 평가하였다. 비교를 위해 범주 변화에 가장 일반적인 방법인 범주 변화가 발생한 노드에서 트리를 재생성하는 경우와 비교하였다. 기본 트리 생성 알고리즘으로 BOAT를 이용하였으며 BOAT를 사용한 이유는 BOAT가 신뢰구간 및 순차적 트리 생성을 지원하며 기존의 방법 중 가장 우수한 성능을 보이기 때문이다. 실험은 (그림 12)와 같이 데이터의 개수를 1백만 데이터에서 5백만까지 1백만씩 증가시키면서 수행하였으며 전역적 범주화를 수행한 후 "전역적 기반 BOAT"는 범주 변화에 대해 본 논문에 제안하는 방법으로 처리한 후 이후 트리 생성을 BOAT가 이용하는 방법으로 수행하였으며 "범주 변경 노드에서 트리 재생성"은 범주 변화가 발생한 노드에서 BOAT가 이용하는 방법으로 트리를 생성하였다.

(그림 12)를 보면 본 논문에서 제안하는 방법이 트리를 재생성하는 방법보다 좋은 성능을 보였다. 그 이유는 제안하는 방법은 범주 변화를 처리하기 위해 유지하고 있는 데이터를 한번 읽는 반면에 트리 재생성 방법은 BOAT의 경우 샘플링 및 검증 과정에서 각 한번씩 최소 두번 데이터를 읽어야 한다. 또한, 함수-6이 함수-9보다 보다 우수한 성능을 보이는 이유는 범주 변화 네가지 경우 중 두 신뢰구간 사이에 새로운 분할 포인트가 발생한 경우가 함수-6에 비해 함수-9의 경우가 더 많았기 때문이다.

네 번째 실험은 대용량 데이터에 대해 본 논문에서 제안하는 전역적 범주화 기반 순차적 방법과 기존의 지역적 범주화 기반 순차적 기법을 BOAT를 이용하여 비교 평가하였으며 실험은 (그림 13)과 같이 데이터의 개수를 1백만 데이터에서 5백만까지 1백만씩 증가시키면서 수행하였다.

(그림 13)에서 전역적 범주화를 이용한 방법이 기존의 지역적 범주화 기반 BOAT보다 좋은 성능을 보였다. 그 이유는 전역적 범주화를 이용할 경우 분할 포인트를 결정하기 위해 한번만 정렬을 수행하는 반면에 지역적 범주화를 이용하면 해당 노드마다 정렬을 수행했기 때문이다. (그림 13)에서 보면, 전역적 범주화 시간은 새로운 데이터 추가에 관계없이 거의 일정한 시간을 보였지만 지역적 범주화 시간은 데이터 수가 증가함에

따라 선형적으로 증가하였다. 특히, 부트스트래핑을 이용하여 샘플 트리를 생성하지 못할 경우 전역적 범주화에 기반한 BOAT는 노드에 유지하고 있는 클래스 분포를 이용하여 분할 변수를 결정할 수 있지만 지역적 범주화에 기반한 BOAT는 RainForest와 같이 AVC-set을 생성해야하며 이는 그 노드에 존재하는 모든 수치형 데이터에 대해 정렬을 수행하기 때문에 많은 시간이 소요된다.

6. 결론 및 향후 연구

본 논문은 대용량 데이터를 처리하기 위해 전역적 범주화 기법을 이용하여 트리를 생성하고 새로운 데이터가 추가될 경우 이를 순차적으로 재생성하는 방법을 제안하였다. 새로운 데이터가 추가될 경우 순차적으로 트리를 재생성하기 위해서는 첫째, 이 데이터가 반영된 범주를 재생성해야하며 둘째, 범주 변화에 맞게 트리의 구조를 변화시켜야한다. 본 논문에서는 효율적인 범주 재생성을 위해 유지하고 있는 기존 빈즈와 병합 없이 직접 다중 빈즈로 부터 범주화를 수행하는 방법 및 구간에 대한 예측을 보정하는 방법 등을 제안하였으며 실험을 통해 기존의 방법에 비해 효율적으로 범주를 재생성함을 보였다. 또한, 범주 변화에 맞는 트리를 재생성하기 위해 분할 포인트 신뢰구간과 트리 재구조화기법을 제안하였으며 실험을 통해 기존의 방법과 비교하였다.

향후 연구로는 첫째, 전역적 범주화에서 샘플 분할 포인트로 정의된 구간의 하위 경계값을 예측하기 위해 필요한 통계치 벡터의 개수 축소에 관한 연구이다. 본 논문의 접근 방법에서 특정 구간의 하위 경계값을 예측하기 위해서는 k개의 클래스가 있을 때, 2^k개의 가상의 통계치 벡터를 유추하고 이들에 평가함수를 적용하여 그 중 최소값을 그 구간에 대한 하위 경계값으로 예측하였다. 이와 같은 접근 방법은 클래스 개수, k가 많게 되면 평가해야할 통계치 벡터가 많아지기 때문에 세그먼트를 이용하는 기존의 방법보다 비효율적이다. 둘째, 신뢰구간의 정확도를 증가시키는 것이다. 새로 생성된 분할 포인트가 신뢰구간내에 존재하면 신뢰구간내에 있는 데이터만 분할함으로써 새로운 범주를 트리에 적용할 수 있다. 따라서, 새로운 분할 포인트가 신뢰구간내에 많이 존재할 수 록 빠른 시간내에 순차적으로 트리를 생성할 수 있다. 셋째, 본 논문에서 사용한 순차적 범주 생성과 범주 변화 처리 기법을 범주화를 사용하는 다른 알고리즘에 적용할 수 있도록 확장하는 것이다. 본 논문에서는 순차적 트리 생성에 적용했지만, 이를 다른 순차적 알고리즘을 사용하는 베이시안 네트웍 및 군집화 알고리즘에 적용할 수 있을 것이다.

참 고 문 헌

[1] J. Cattlet. *MegaInduction: Machine Learning on Very Large Databases*. PhD thesis, University of Sydney, 1991.

[2] P. K. Chan and S. J. Stolfo. Meta-Learing for Multistrategy and parallel leaning. In Proc. *Second Intl. Workshop on Multistrategy Leaning*, pp.150-165, 1993.

[3] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. *Proceedings of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, 1996.

[4] J, Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. *Proceedings of Very Large DataBase(VLDB)*, 1996.

[5] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest - A framework for fast decision tree construction of large datasets. VLDB 1996.

[6] P. E Utgoff. Incremental induction of decision trees. *Machine Learning*, Vol.4, pp.161-186, 1989.

[7] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh. Boat-optimistic decision tree construction. *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1999.

[8] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous Features. *Proceedings of Twelfth International Conference on Machine Learning*, pp.194-202, 1995.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[10] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(8), pp.1-106, 1986.

[11] U. M. Fayyad, K. B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, Vol.8, pp.87-102, 1992.

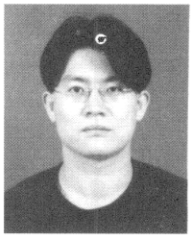
[12] U. M. Fayyad, K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning, Proceedings of the 13th International Joint Conference on Artificial Intelligence, *Morgan Kaufmann*, pp.1022-1027.

[13] T. Elomaa and J. Rousu. General and efficient multisplitting of numerical attributes. *Machine Learning*, Vol.36, pp.200-244, 1999.

[14] T. Elomaa and J. Rousu. Generalizing boundary points. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, Menlo Park, CA, 2000. AAAI Press. In press.

[15] 한경식, 이수원. 전역적 범주화를 위한 샘플 포인트를 이용한 점진적 기법. 정보과학회 논문지 : 소프트웨어 및 응용, 제31권, 제7호, pp.849-858. 2004. 07.

[16] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Database mining: A performance perspective. IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.6, pp.914-952, 1993.



한 경 식

e-mail : hanksjang@empal.com

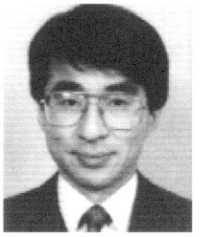
1997년 숭실대학교 공과대학 인공지능학과(학사)

1999년 숭실대학교 정보과학대학 컴퓨터학과(석사)

2002년 숭실대학교 정보과학대학 컴퓨터학과 박사수료

2002년~현재 (주)인우기술 선임연구원

관심분야 : 인공지능, CRM, 데이터마이닝



이 수 원

e-mail : swlee@computing.ssu.ac.kr

1982년 서울대학교 계산통계학과(학사)

1984년 한국과학기술원 전산학과(석사)

1994년 U. of Southern California 전산학과(박사)

1995~현재 숭실대학교 컴퓨터학부 부교수

관심분야 : 인공지능, 데이터마이닝, 에이전트, 기계학습